ΗΥ 232
Οργάνωση και Σχεδίαση Υπολογιστών

Διάλεξη 12
Καθυστερήσεις (Stalls)
Εκκενώσεις Εντολών (Flushing)

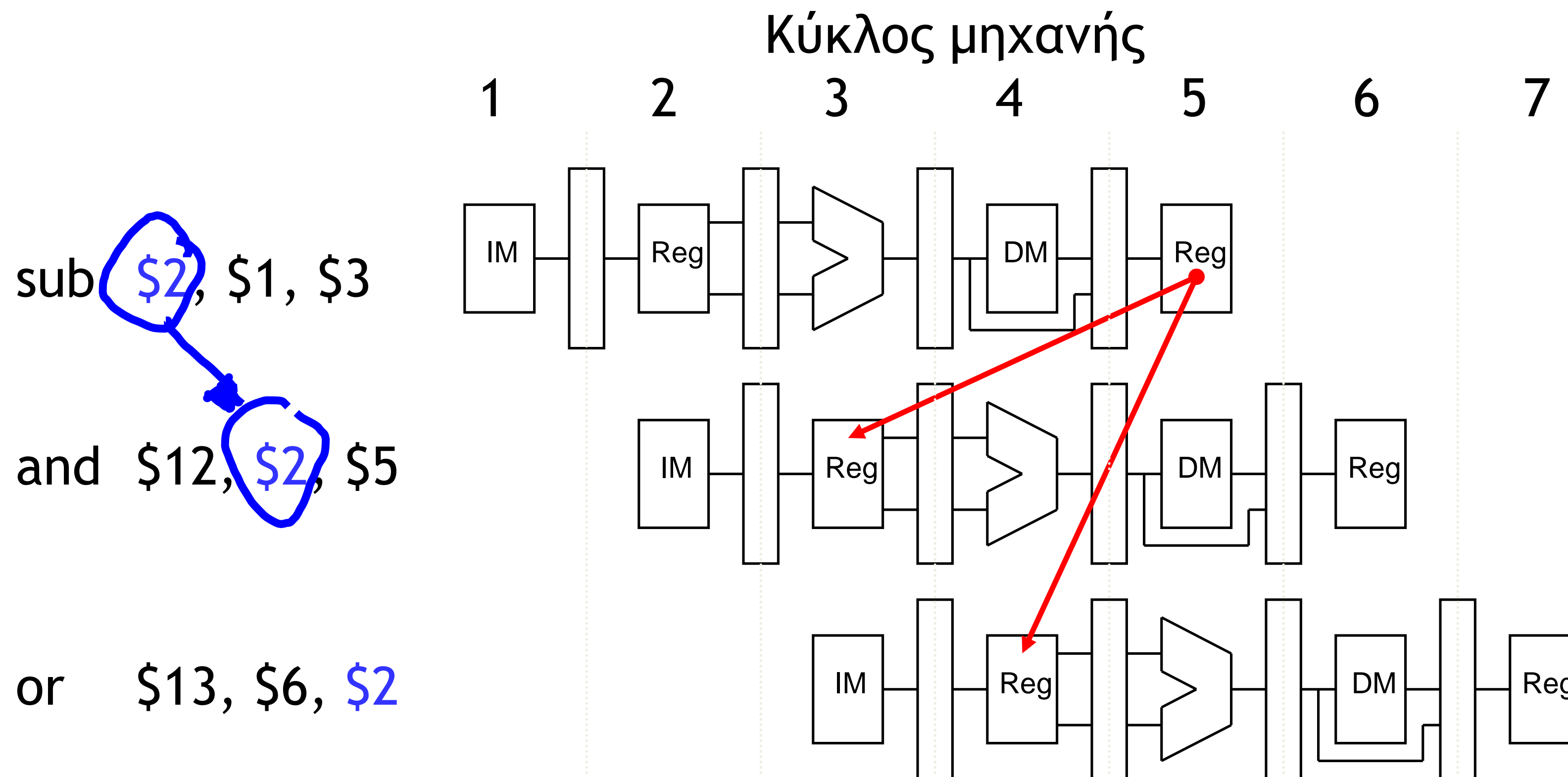
Νίκος Μπέλλας
Τμήμα Μηχανικών Η/Υ, Τηλεπικοινωνιών και Δικτύων

Καθυστερήσεις και Εκκενώσεις Εντολών

- Οι δομικοί κίνδυνοι (data hazards), όπως είδαμε, συμβαίνουν όταν εντολές εξαρτώνται από άλλες εντολές που βρίσκονται ακόμα προς εκτέλεση στο pipeline.
  - Πολλοί τέτοιοι κίνδυνοι μπορούν να επιλυθούν μέσω της προώθησης δεδομένων από τους καταχωρητές διοχέτευσης στην ALU; Αντί να περιμένουμε τα δεδομένα αυτά να γραφούν πρώτα στους καταχωρητές .
  - Το πλεονέκτημα είναι ότι η προώθηση επιτρέπει ο επεξεργαστής να τρέχει με πλήρη ταχύτητα.
- Σήμερα θα δούμε κάποια πραγματικά προβλήματα στην μικρο-αρχιτεκτονική διοχέτευσης που επηρεάζουν την ταχύτητα του επεξεργαστή .
  - Η προώθηση από εντολές load δεν είναι πάντα αποδοτικές.
  - Οι εντολές διακλάδωσης επηρεάζουν την εκτέλεση των επόμενων εντολών.
- Σε αυτές τις δύο περιπτώσεις θα πρέπει ίσως να καθυστερήσουμε την εκτέλεση του προγράμματος ή να εκκενώσουμε μέρος του pipeline από εκτελούμενες εντολές

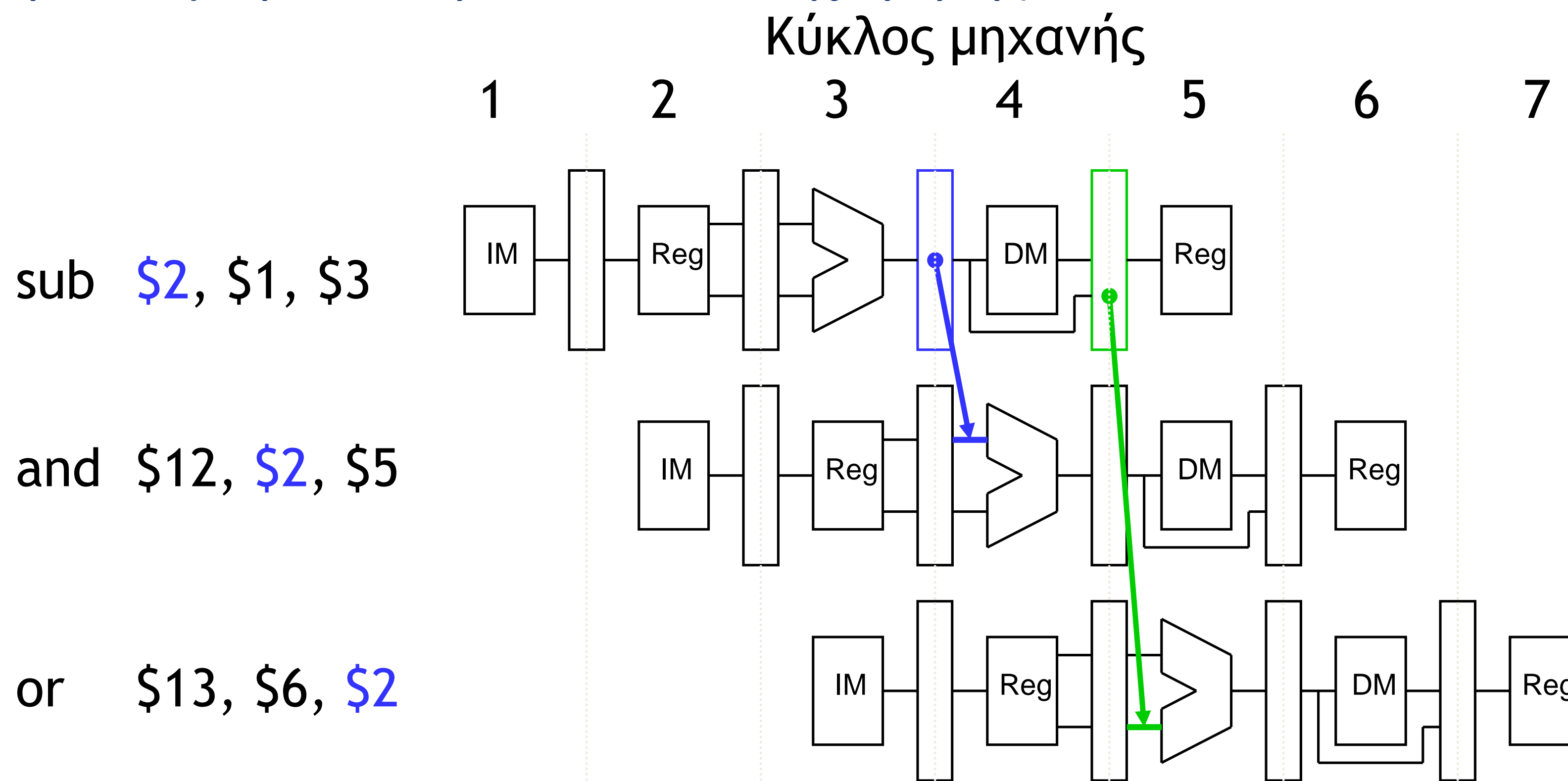
Κίνδυνος Δεδομένων (Data Hazard)

- Κίνδυνος δεδομένων συμβαίνει εάν μια εντολή χρειάζεται δεδομένα που δεν είναι ακόμα έτοιμα στους καταχωρητές.
  - Οι εντολές **`and`** και **`or`** χρειάζονται τον καταχωρητή $2 στον 3ο και 4ο κύκλο, αντίστοιχα.
  - Όμως ο $2 γράφεται μόλις στον 5ο κύκλο από την εντολή **`sub`**.



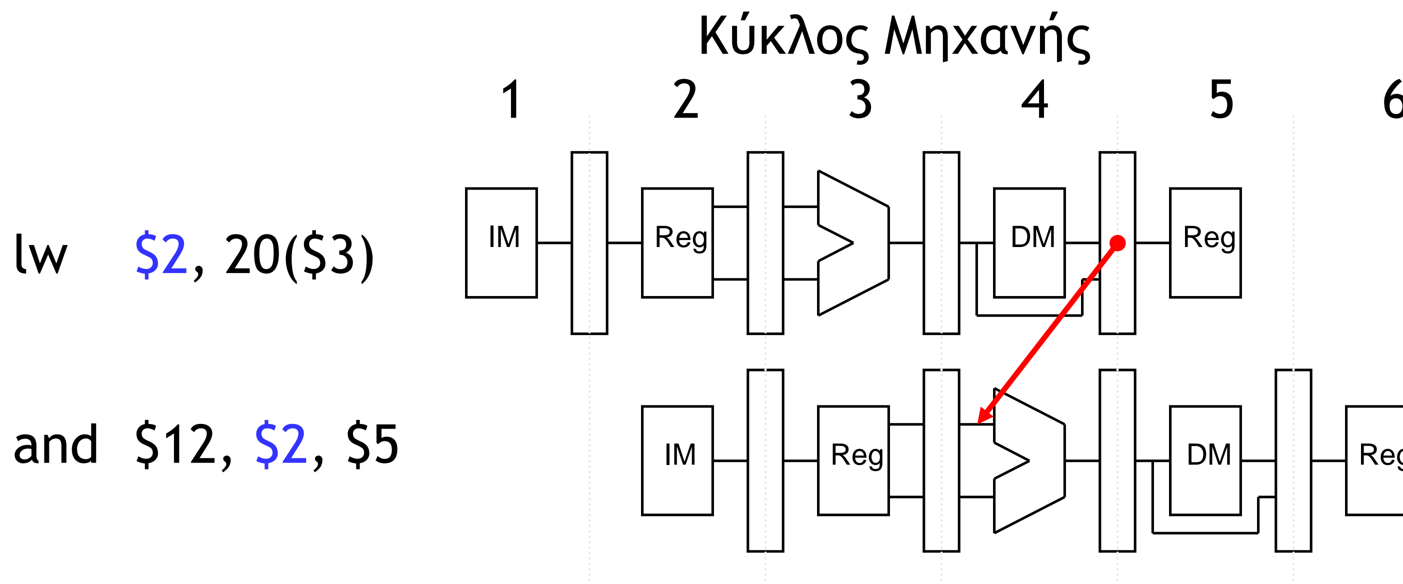
Προώθηση Δεδομένων

- Η τελική τιμή του καταχωρητή $2 ($1 - $3) έχει ήδη υπολογιστεί από τον κύκλο 3 (στάδιο EX) , απλά δεν έχει γραφεί ακόμα στον $2.
- Η προώθηση δεδομένων επιτρέπει δεδομένα να τροφοδοτούνται μέσω των καταχωρητών διοχέτευσης κατευθείαν στην ALU χωρίς την ανάγκη να διαβαστεί ο καταχωρητής.



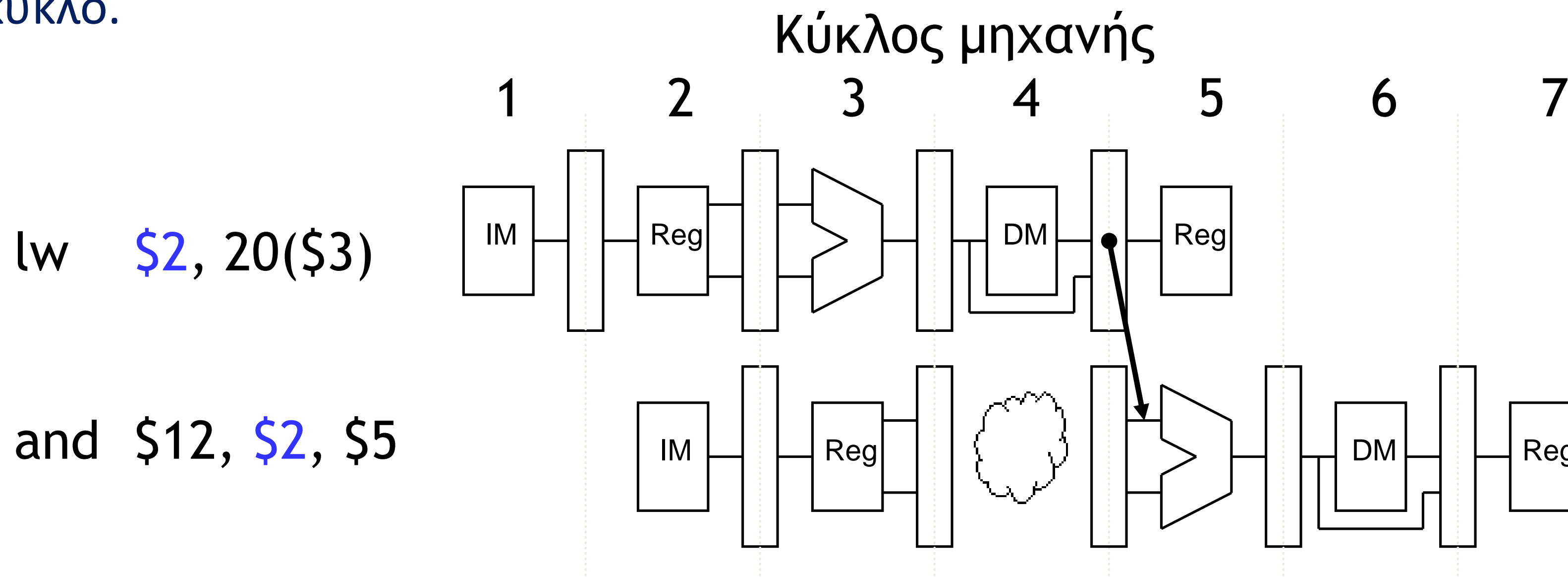
Εντολές load?

- Τι θα συνέβαινε εάν η πρώτη εντολή ήταν **`lw`**;
  - Η εντολή **`lw`** παράγει δεδομένα στο τέλος του κύκλου 4 στο στάδιο MEM
  - Αλλά η **`and`** χρειάζεται αυτά τα δεδομένα στην αρχή του ίδιου κύκλου!
- Αυτό είναι ένας πραγματικός κίνδυνος δεδομένων. Δεν μπορούμε να προωθήσουμε τα δεδομένα μέσω των καταχωρητών διοχέτευσης, όπως πριν.



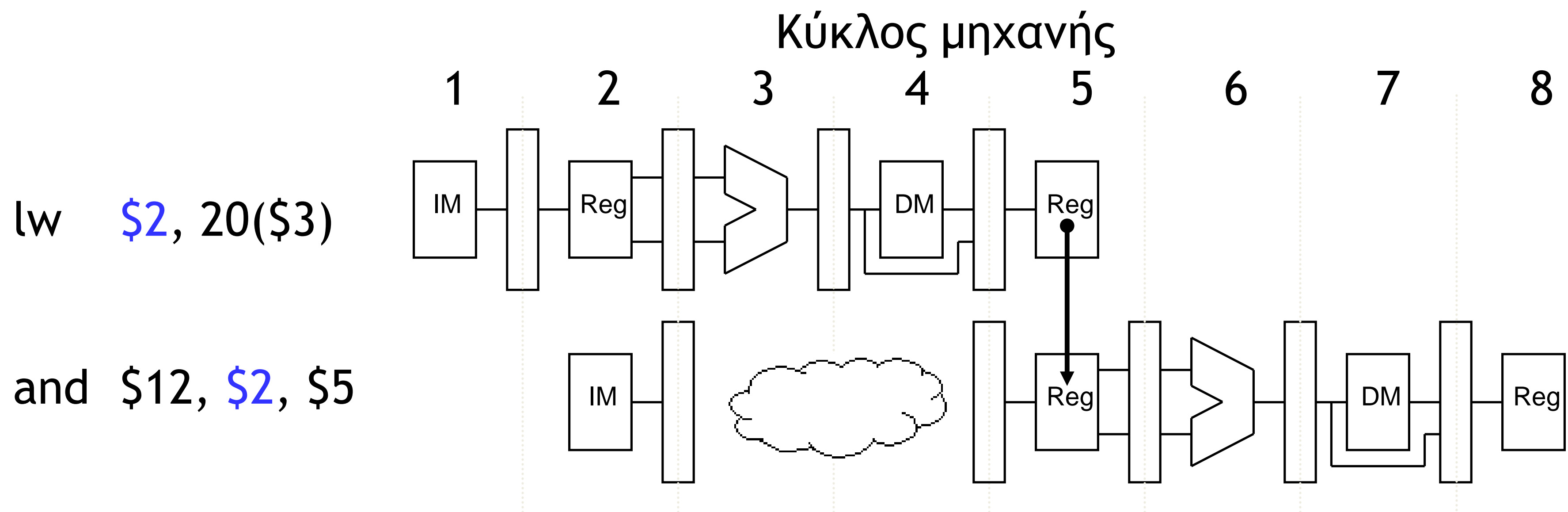
Καθυστέρηση (Stall)

- Η λύση είναι να καθυστερήσουμε το pipeline για έναν κύκλο
- Μπορούμε να καθυστερήσουμε την εντολή **and** με το να τοποθετήσουμε μία φυσαλίδα (bubble) ενός κύκλου.
- Αυτό θα καθυστερήσει την **and** και όλες τις εντολές μετά από αυτήν κατά ένα κύκλο.



- Μετά από αυτό, χρησιμοποιούμε τον καταχωρητή διοχέτευσης MEM/WB για να στείλουμε τα δεδομένα που διαβάζει η **lw** από την μνήμη στην ALU.

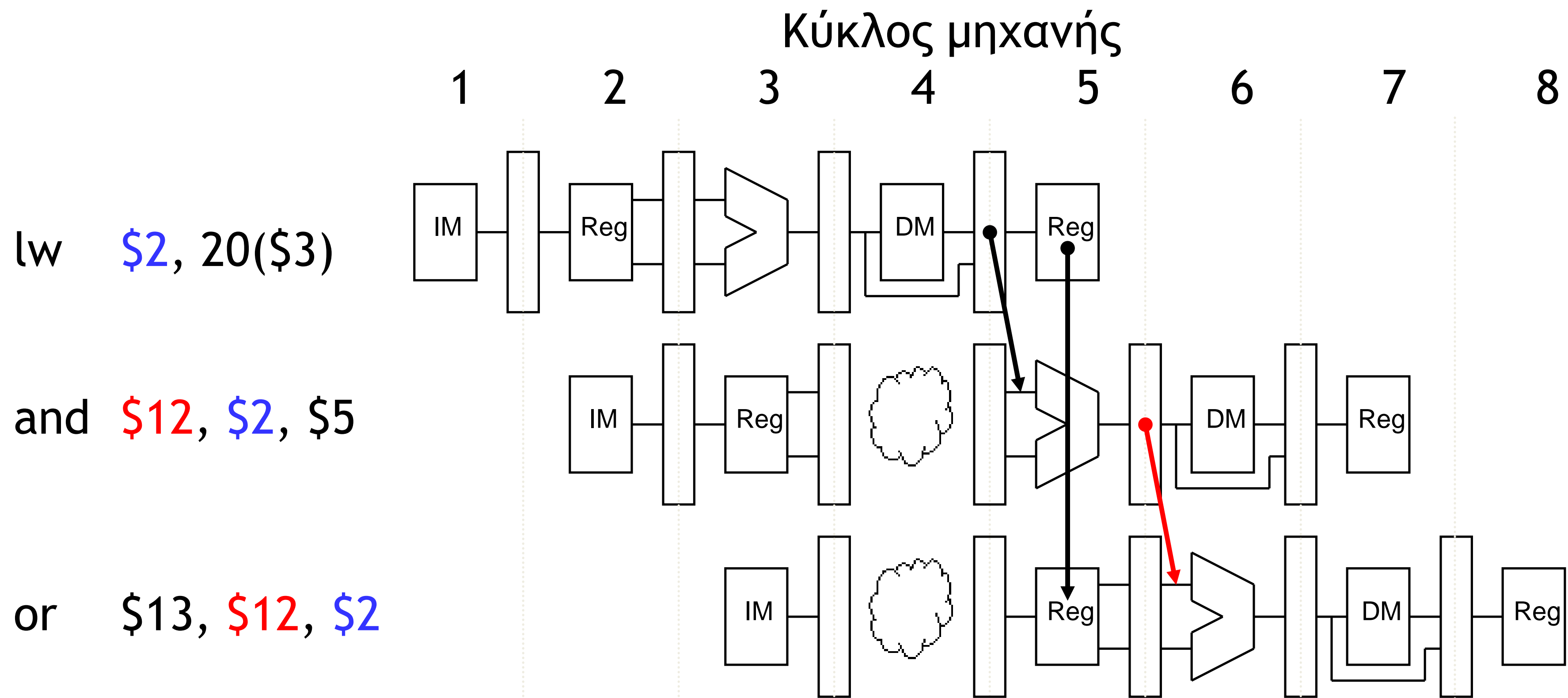
Καθυστέρηση και Προώθηση



- Η προώθηση μειώνει τον χρόνο καθυστέρησης από 2 σε 1 κύκλο μηχανής

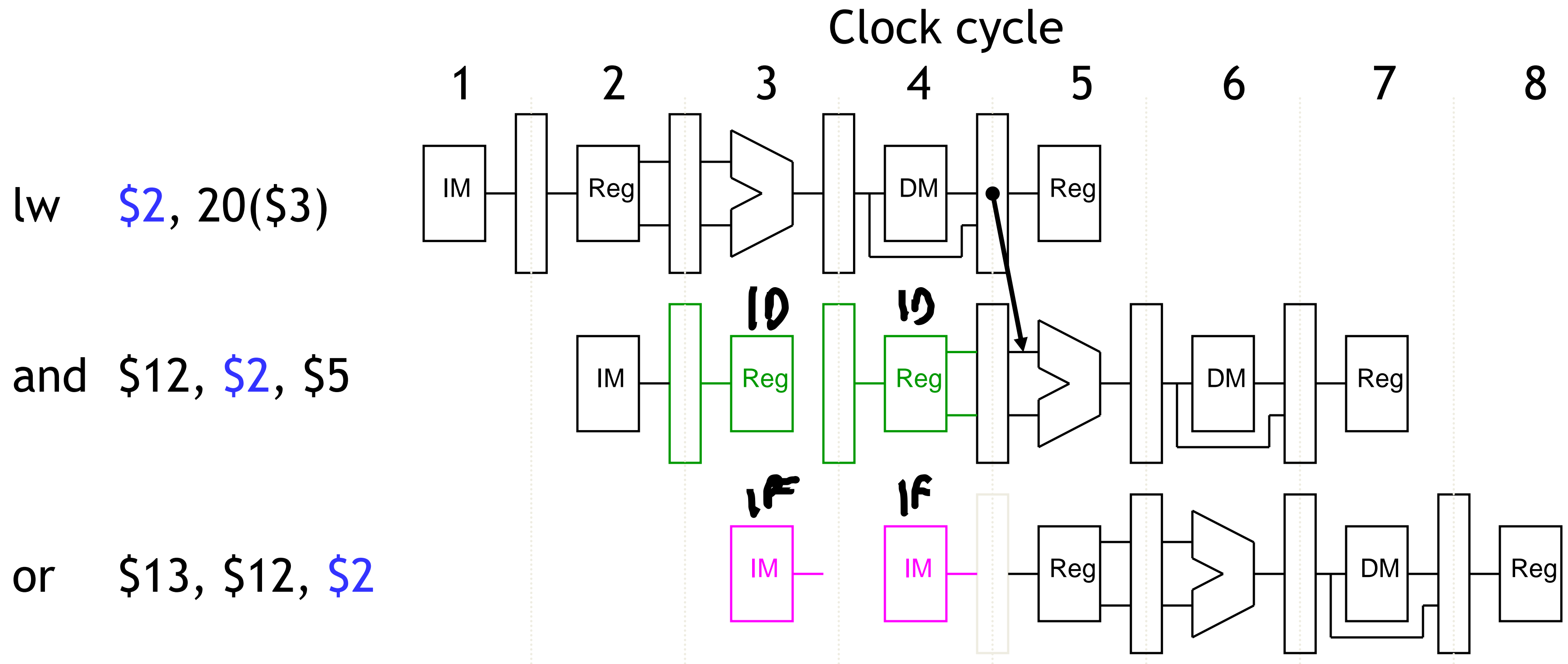
Η καθυστέρηση αφορά όλο το pipeline

- Η καθυστέρηση της εντολής **and** σημαίνει και την καθυστέρηση και της **or**. Καθώς και όλων των άλλων εντολών που ακολουθούν.
  - Δεν μπορούμε να έχουμε δύο εντολές στο ίδιο στάδιο του pipeline.
  - Δεν μπορεί επίσης η **or** να ξεπεράσει την **and** και να τελειώσει πριν από αυτήν



Υλοποίηση καθυστερήσεων

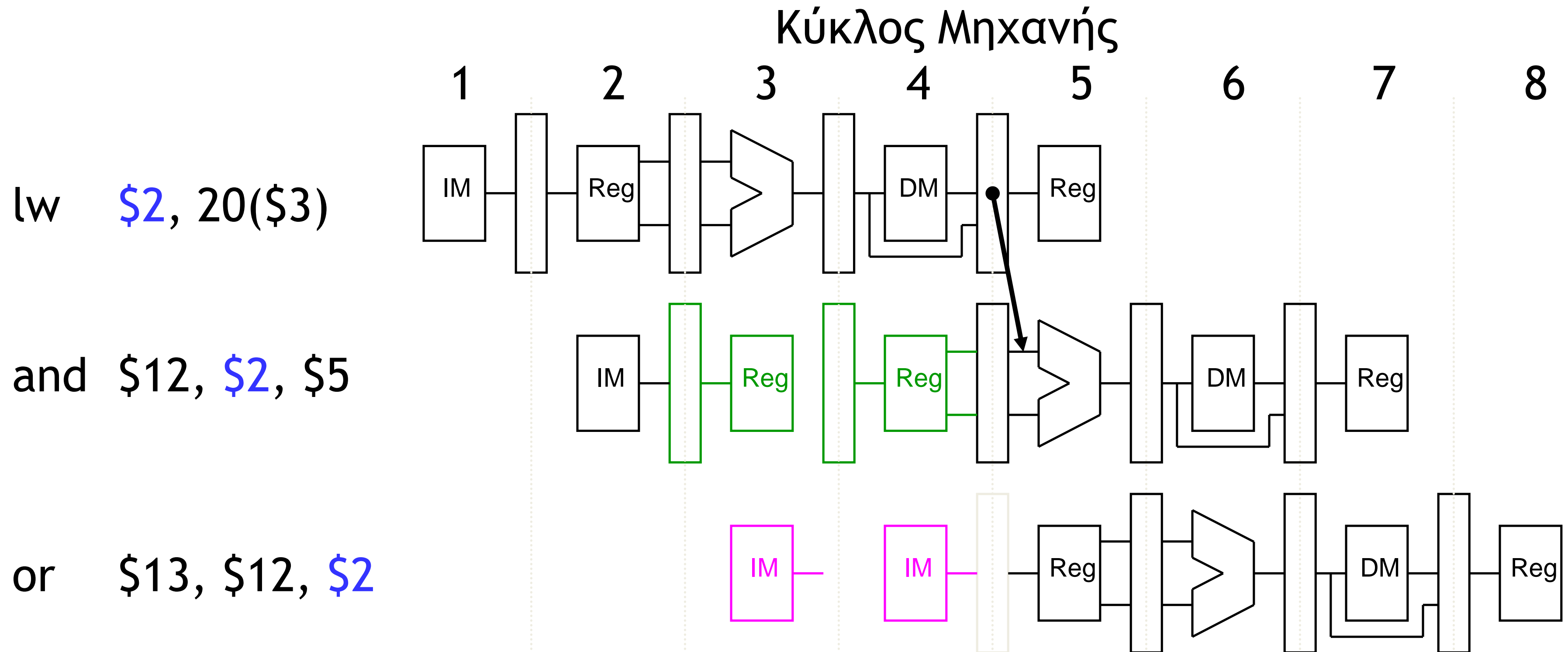
- Υλοποιούμε την καθυστέρηση (stall) με το να μπλοκάρουμε τις εντολές που είναι στα σταδία ID και IF.



- Αυτό υλοποιείται με το να:
  - αφήσουμε την τιμή του PC την ίδια, ώστε να μην διαβαστεί άλλη εντολή.
  - να μην αλλάξουμε την τιμή του καταχωρητή IF/ID, ώστε να μην αλλάξει η εντολή στο στάδιο ID.

Στάδια EX, MEM, WB

- Μένει μόνο να δούμε πως θα υλοποιηθεί η φυσαλίδα. Δηλαδή το στάδιο EX στον κύκλο 4, το στάδιο MEM στον κύκλο 5, και το στάδιο WB στον κύκλο 6.

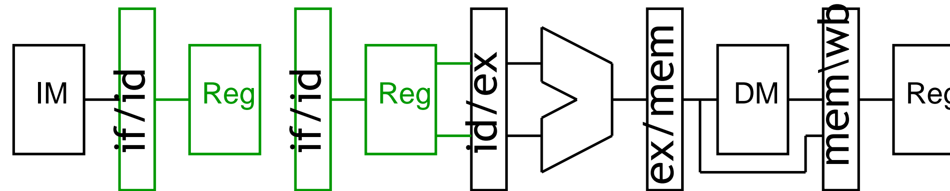


- Η φυσαλίδα (bubble) υλοποιείται με το να θέσουμε σε αυτό το στάδιο την εντολή να είναι **nop**. Με άλλα λόγια να θέσουμε τα σήματα ελέγχου στο 0, για να αποκλείσουμε να γίνει κάποια εγγραφή σε καταχωρητή ή μνήμη κατά λάθος.

Ανίχνευση Καθυστερήσεων

lw $2, 20($3)

and $12, $2, $5



- Η ανίχνευση της καθυστέρησης γίνεται όταν η εντολή **`lw`** είναι στο EX στάδιο και η επόμενη εξαρτώμενη εντολή (**`and`**) είναι στο ID στάδιο.
- Καθυστέρηση υπάρχει όταν η εντολή στο EX στάδιο είναι LOAD

`ID/EX.MemRead = 1`

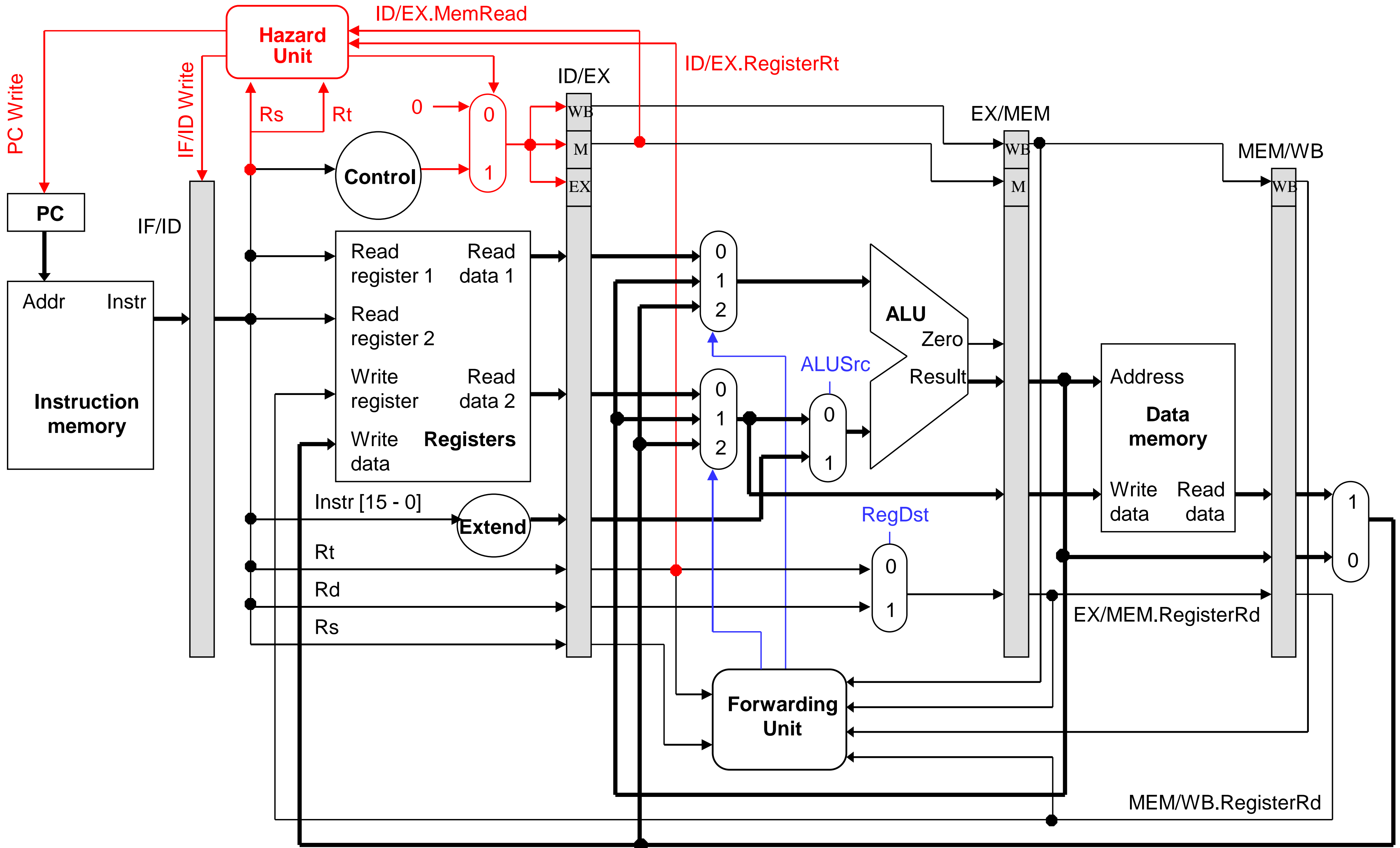
και ο προορισμός του LOAD είναι ένας από τους καταχωρητές εισόδου της επόμενης εντολής (στο ID στάδιο).

```
ID/EX.RegisterRt = IF/ID.RegisterRs
                        or
ID/EX.RegisterRt = IF/ID.RegisterRt
```

- Συνοπτικά:

```
if (ID/EX.MemRead = 1 and
   (ID/EX.RegisterRt = IF/ID.RegisterRs or
    ID/EX.RegisterRt = IF/ID.RegisterRt))
              then stall
```

Η μονάδα ελέγχου κινδύνων (Hazard Unit) είναι στο στάδιο ID

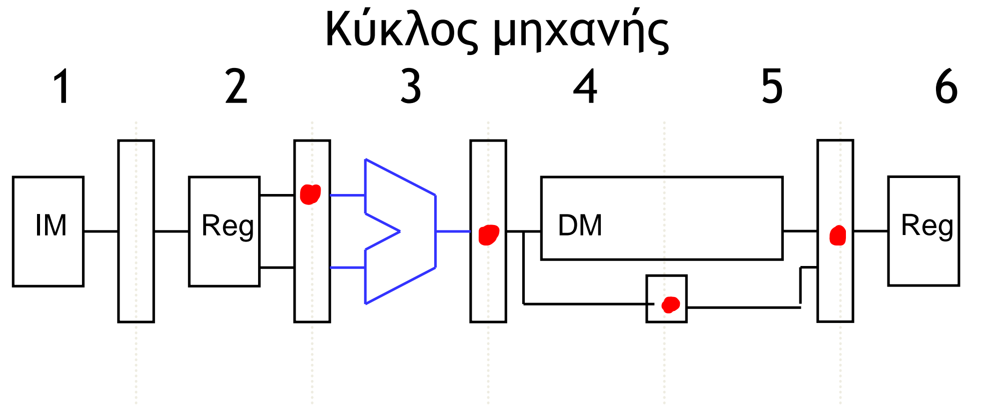


Μονάδα ελέγχου κινδύνων (Hazard Unit)

- Το *stall* της προηγούμενης εξίσωσης είναι η δημιουργία τριών επιπλέον σημάτων.
  - Δύο νέα σήματα ελέγχου *PCWrite* και *IF/ID Write* που ουσιαστικά είναι *Write Enable* σήματα του *PC* και του καταχωρητή *IF/ID,* αντίστοιχα.
  - Ένα *mux select* σήμα σε έναν νέο καταχωρητή ο οποίος θέτει όλα τα σήματα ελέγχου του ID/EX στο 0, όταν γίνει ανίχνευση καθυστέρησης. Εδώ ουσιαστικά γίνεται η δημιουργία της εντολής **`nop`**.

Να μια καλή ερώτηση

- Έστω ότι η προσπέλαση της μνήμης είναι πολύ αργή και χρειάζεται 2 κύκλους μηχανής.

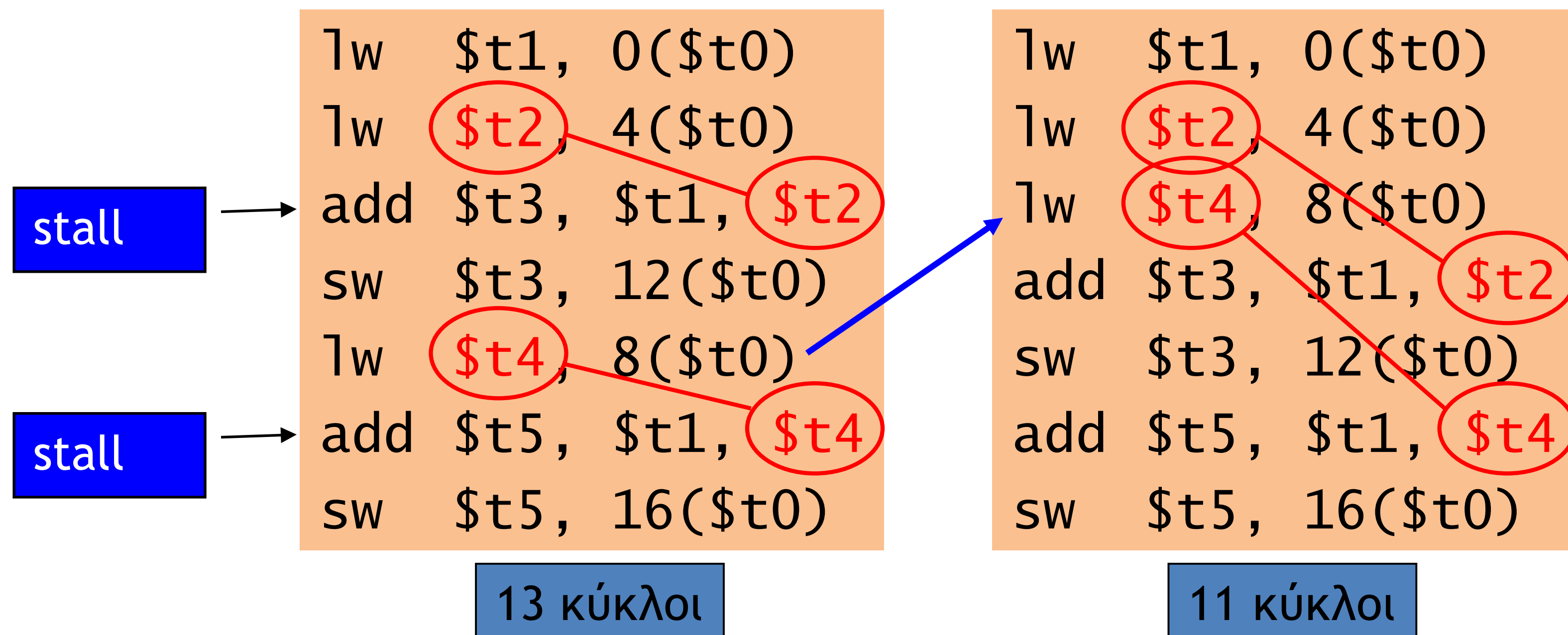


- Ποιες από τις παρακάτω εντολές χρειάζονται προώθηση ή/και καθυστέρηση;
- Πόσες εισόδους θα έχουν οι πολυπλέκτες στο στάδιο EX;

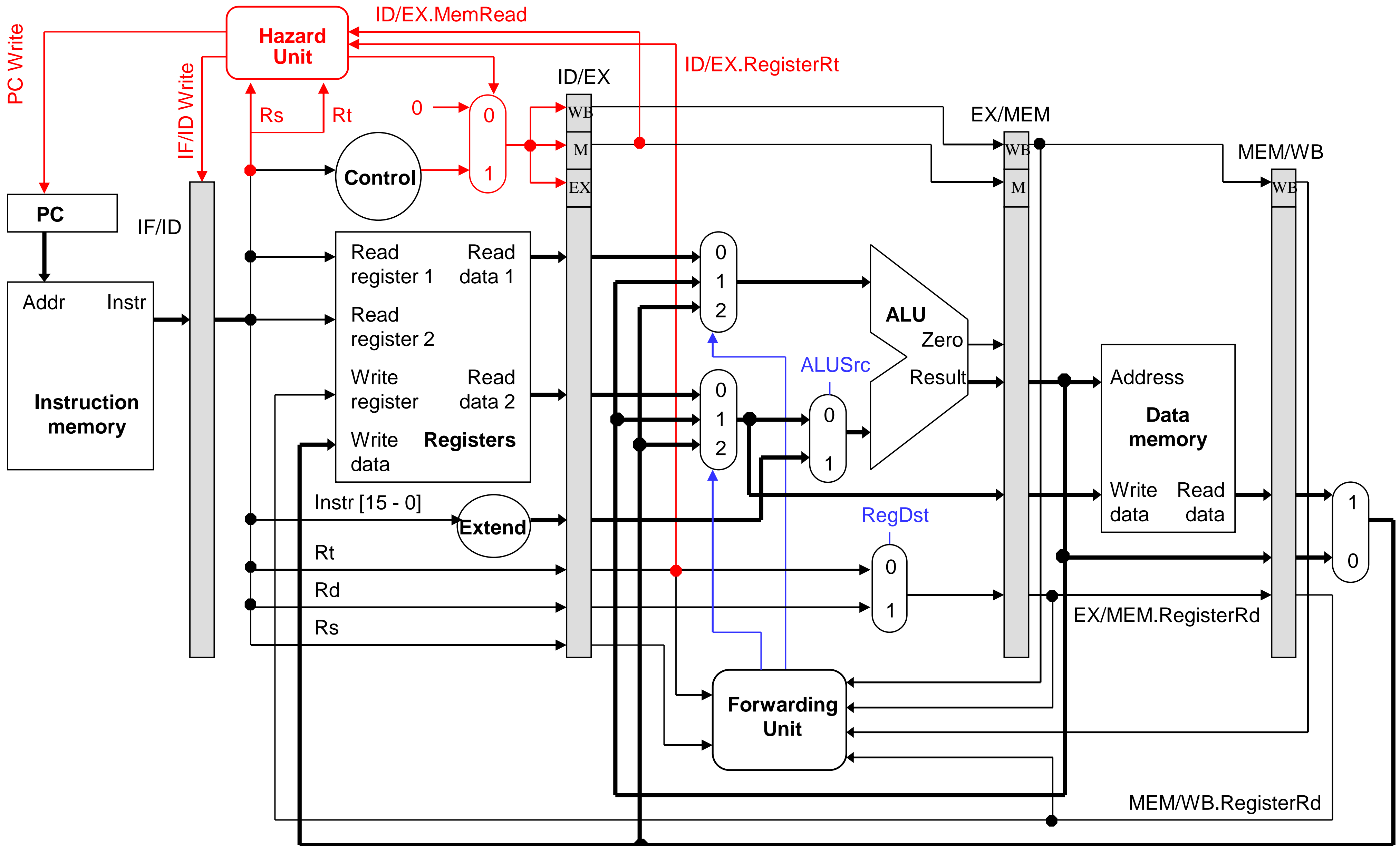
lw r13, 0(r11)	IF	ID	EX	M1	M2	WB				
add r7, r8, r9		IF	ID	EX	M1	M2	WB			
add r15, r7, r13			IF	ID	ID	EX	M1	M2	WB	

Τι μπορεί να κάνει ο Compiler ή ο προγραμματιστής assembly

- Χώρισε τις εντολές LOAD από τις εντολές που εξαρτώνται από αυτές με αρκετές ανεξάρτητες εντολές ώστε να μην υπάρχει πρόβλημα καθυστέρησης
- Κλασσική βελτιστοποίηση του compiler είναι να προγραμματίσει όλες τις εντολές LOAD στην αρχή του κώδικα.

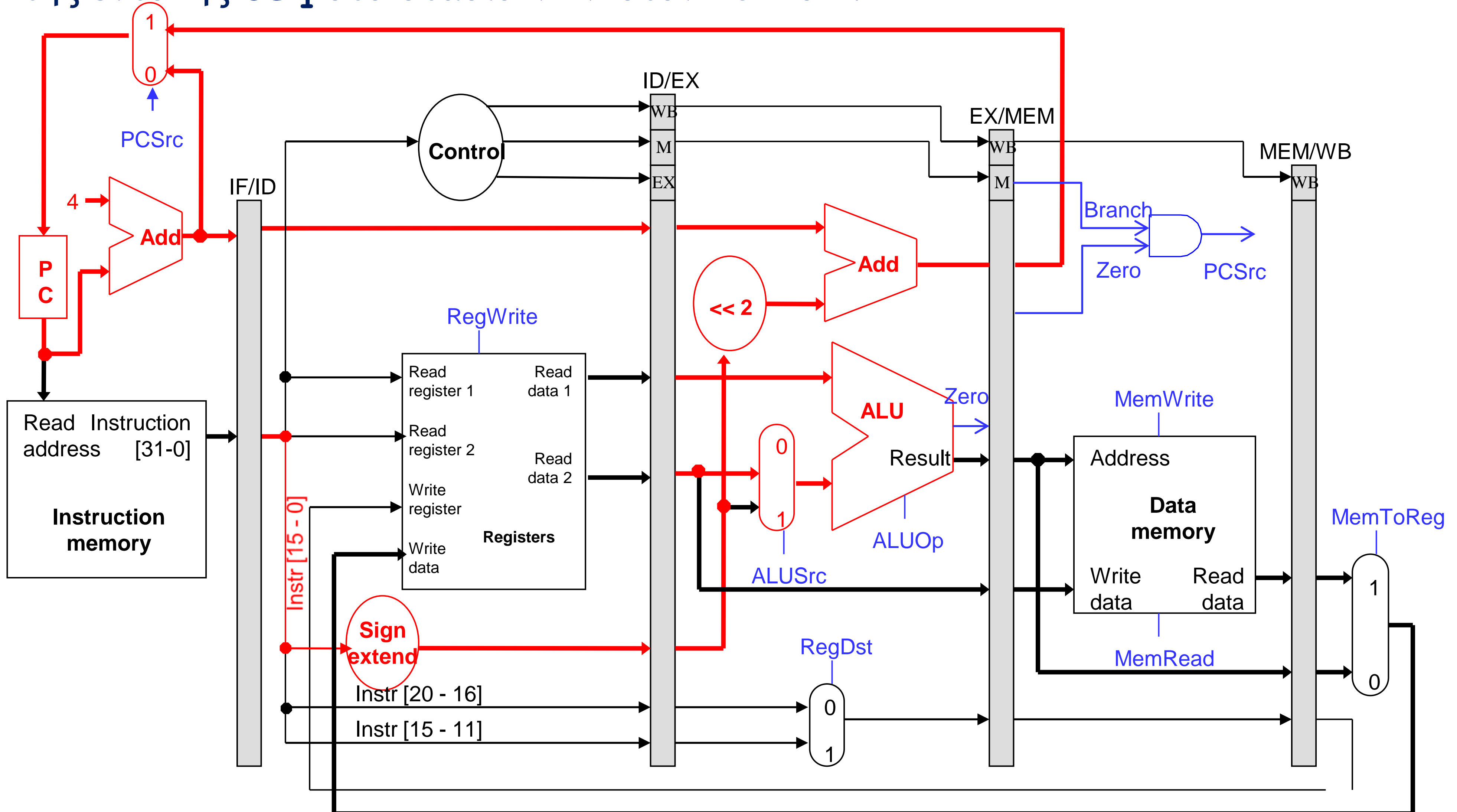


Η μέχρι τώρα μικρο-αρχιτεκτονική (Προώθηση, Καθυστέρηση, Εκκένωση)

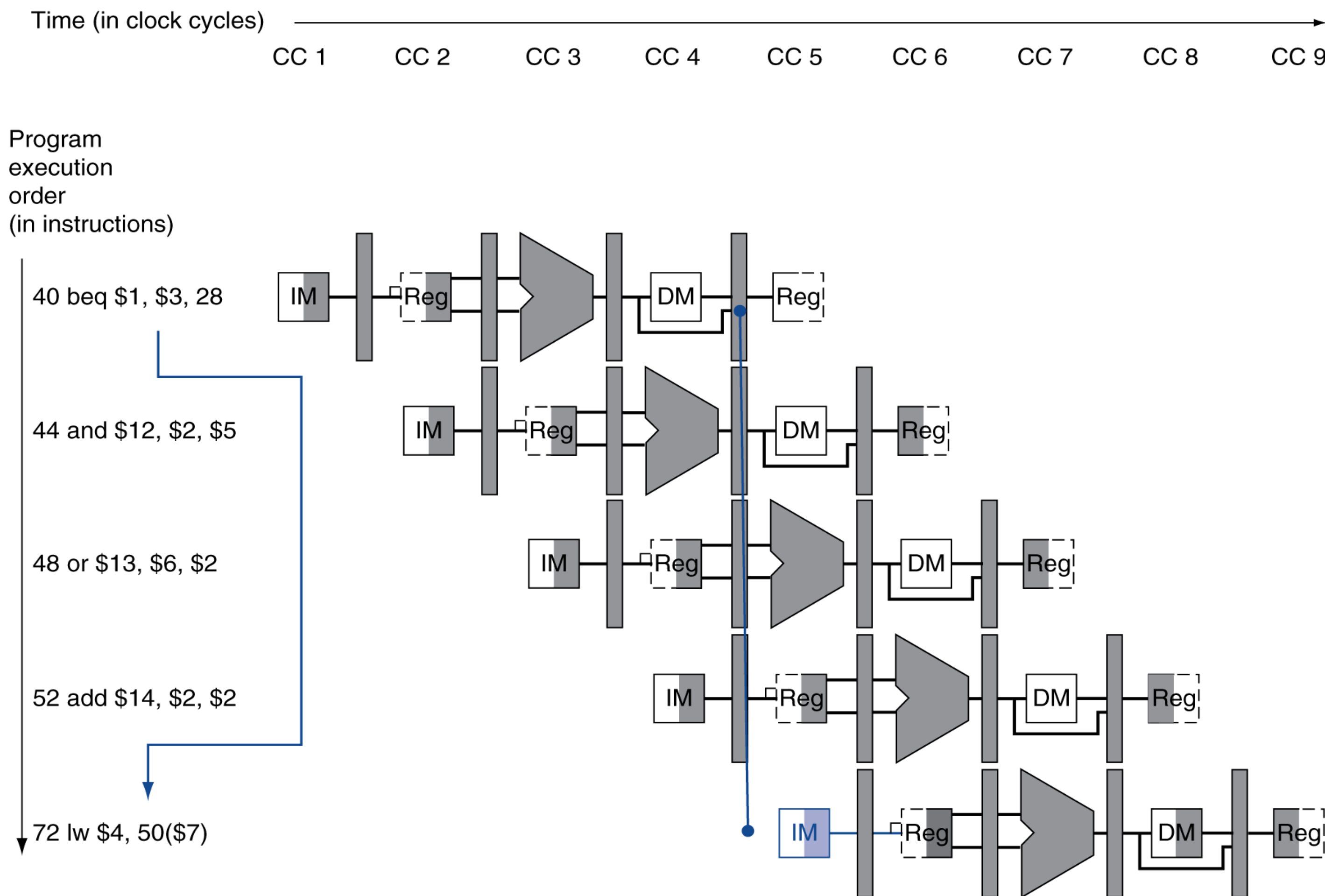


Η μέχρι τώρα μικρο-αρχιτεκτονική (Εντολές Διακλάδωσης)

Η μικρο-αρχιτεκτονική μας παίρνει απόφαση για την κατεύθυνση και τον προορισμό της εντολής `beq` στο στάδιο MEM στον κύκλο 4.



Κίνδυνοι Ελέγχου (Control Hazards)



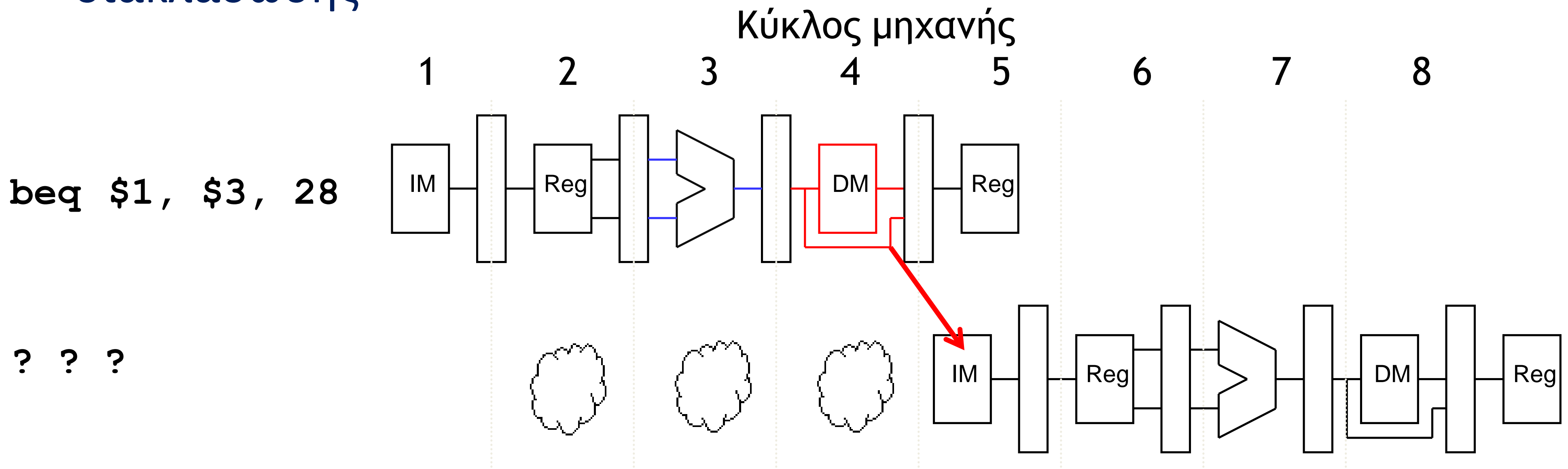
Η μικρο-αρχιτεκτονική μας παίρνει απόφαση για την κατεύθυνση της εντολής **`beq`** στο στάδιο MEM στον κύκλο 4.

Αλλά πρέπει να ξέρουμε αυτήν απόφαση πιο νωρίς για να φέρουμε την σωστή επόμενη εντολή.

Αλλιώς, αντί να φέρουμε την **`lw`** εάν η εντολή **`beq`** είναι TAKEN, θα φέρουμε την **`and`**

Καθυστέρηση (Stalling)

- Μία λύση είναι να καθυστερήσουμε το pipeline κατά 3 κύκλους μέχρι να πάρουμε την απόφαση για το αποτέλεσμα της εντολής διακλάδωσης



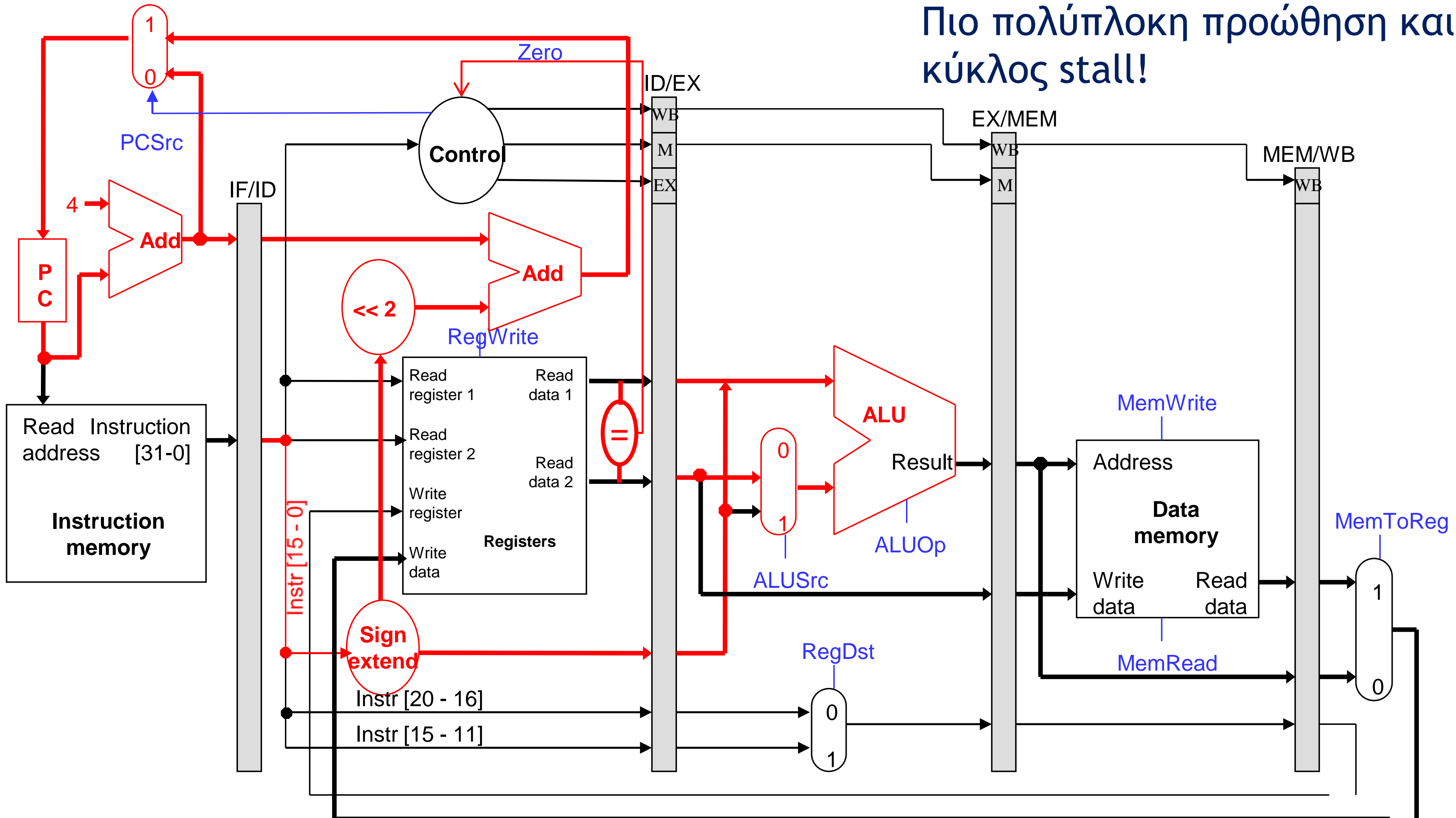
- Στην συγκεκριμένη περίπτωση, θα πρέπει να δημιουργήσουμε 3 εντολές **nop** ώστε στην συνέχεια να φέρουμε την σωστή εντολή από την μνήμη.
- Τρείς κύκλοι κάθε φορά που έχουμε εντολή διακλάδωσης είναι πολλοί. Μπορούμε να τους μειώσουμε;

Μείωση των κύκλων καθυστέρησης

- Η εκτέλεση μιας εντολής διακλάδωσης (**`beq rs, rt, Label`**) απαιτεί δύο πράγματα:
  - Τον υπολογισμό της διεύθυνσης προορισμού: PC <= Label
  - Την απόφαση για το εάν η εντολή διακλάδωσης είναι TAKEN ή NOT TAKEN
- Και οι δύο αυτοί υπολογισμοί πρέπει να γίνουν νωρίτερα (δηλ. να μεταφερθούν πιο πριν στο pipeline) για να μπορέσουμε να μειώσουμε την ποινή της καθυστέρησης. Θα μεταφέρουμε και τους δύο αυτούς υπολογισμούς στο στάδιο ID.
  - Ο υπολογισμός της διεύθυνσης προορισμού μπορεί εύκολα να μεταφερθεί από το στάδιο EX στο στάδιο ID με το να μεταφέρουμε την μικρή ALU και το left shift by 2.
  - Η απόφαση για την διακλάδωση μπορεί να γίνει με επιπλέον hardware που απλά θα ελέγχει εάν οι 2 καταχωρητές **`rs`** και **`rt`** που μόλις διαβάστηκαν από το register file είναι ίσοι.
  - Αυτό γίνεται απλά με 32 πύλες XOR. Μπορεί όμως να αυξήσει την περίοδο του ρολογιού.

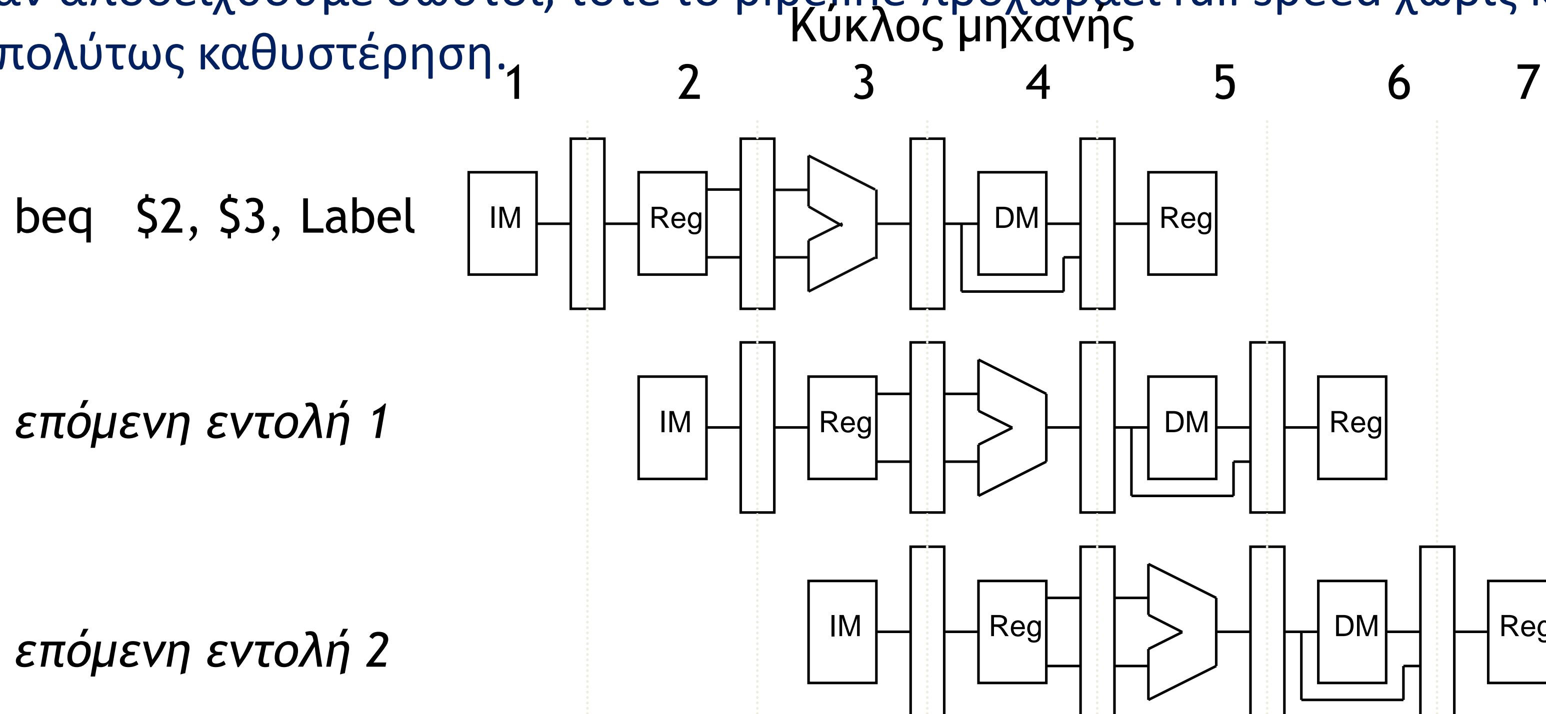
Νέο pipeline για μικρότερο penalty στις εντολές διακλάδωση

Τι θα γίνει εάν έχουμε τις εντολές
sub $2, $1, $3
beq $4, $2, L
Πιο πολύπλοκη προώθηση και ένας κύκλος stall!



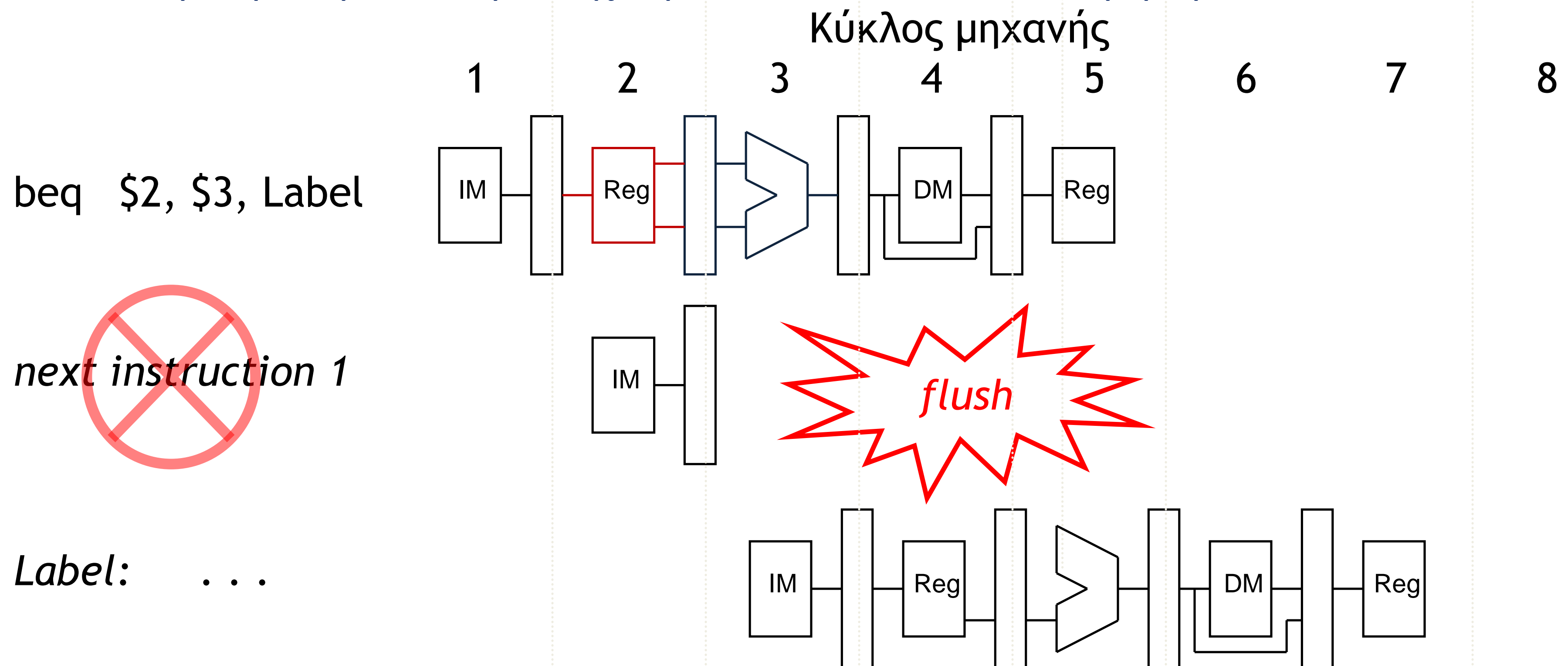
Πρόβλεψη διακλάδωσης

- Αφού μειώσαμε τους κύκλους καθυστέρησης από 3 σε 1 ας δούμε πως μπορούμε να μειώσουμε ακόμα περαιτέρω τους κινδύνους ελέγχου
- Μπορούμε να "μαντέψουμε" την απόφαση της διακλάδωσης με διάφορους τρόπους
  - Η πιο απλή προσέγγιση είναι να θεωρήσουμε ότι η διακλάδωση είναι πάντα NOT TAKEN.
  - Απλά αύξησε τον PC<=PC+4 και φέρε την επόμενη εντολή ως σαν να μην είχαμε διακλάδωση
- Εάν αποδειχθούμε σωστοί, τότε το pipeline προχωράει full speed χωρίς καμία απολύτως καθυστέρηση.



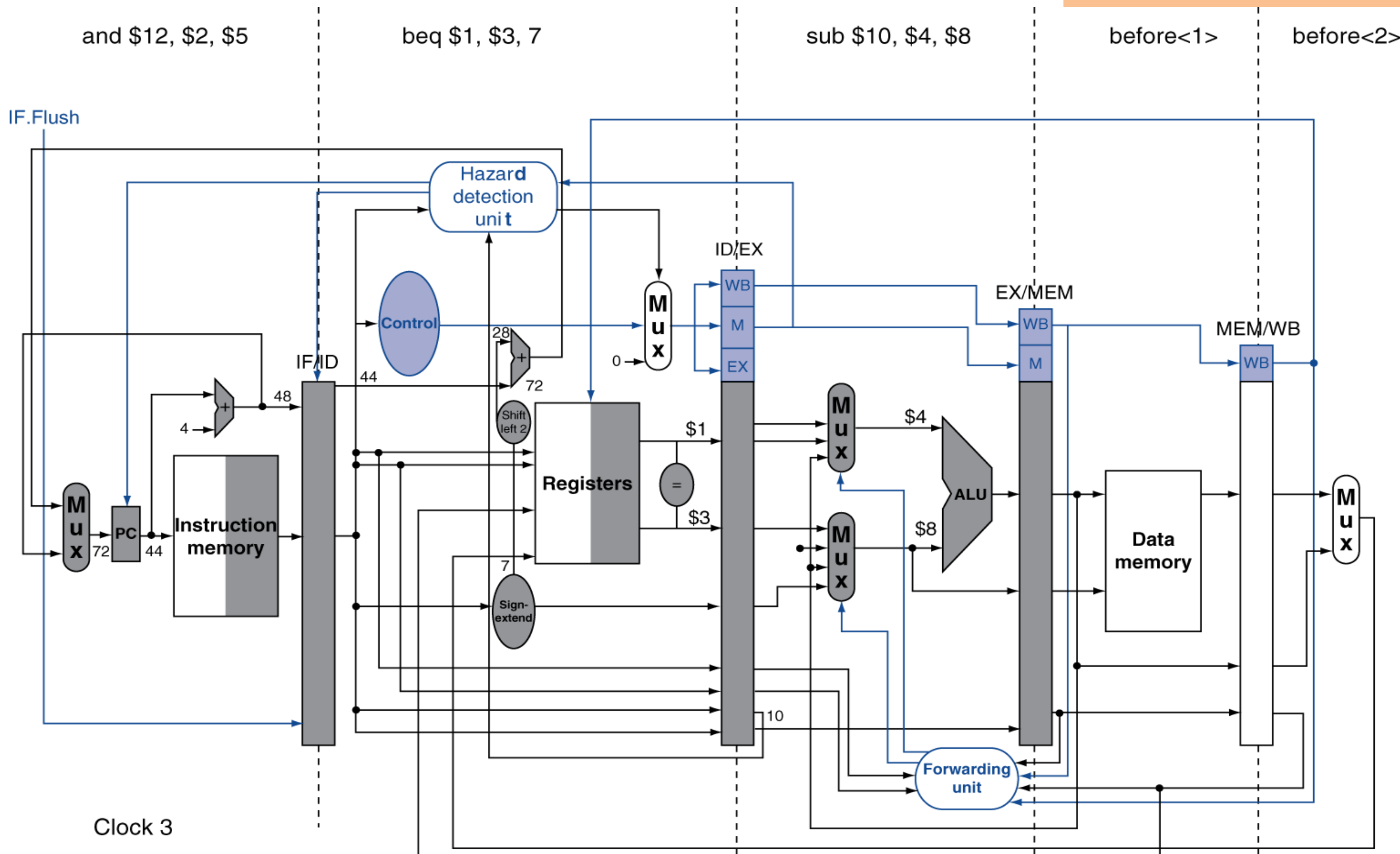
Πρόβλεψη διακλάδωσης

- Εάν η υπόθεσή μας είναι λάθος και η διακλάδωση είναι TAKEN, τότε θα έχουμε φέρει λάθος εντολή.
  1. Θα πρέπει να κάνουμε flush την εντολή αυτή και να βάλουμε στην θέση της μια εντολή **nop**. Το flush γίνεται στον καταχωρητή IF/ID
  2. Θα πρέπει επίσης να αλλάξουμε τον PC ώστε να αρχίσουμε αμέσως μετά να διαβάζουμε εντολές από την διεύθυνση μνήμης Label.
- Σε αυτήν την περίπτωση θα έχουμε έναν κύκλο καθυστέρηση

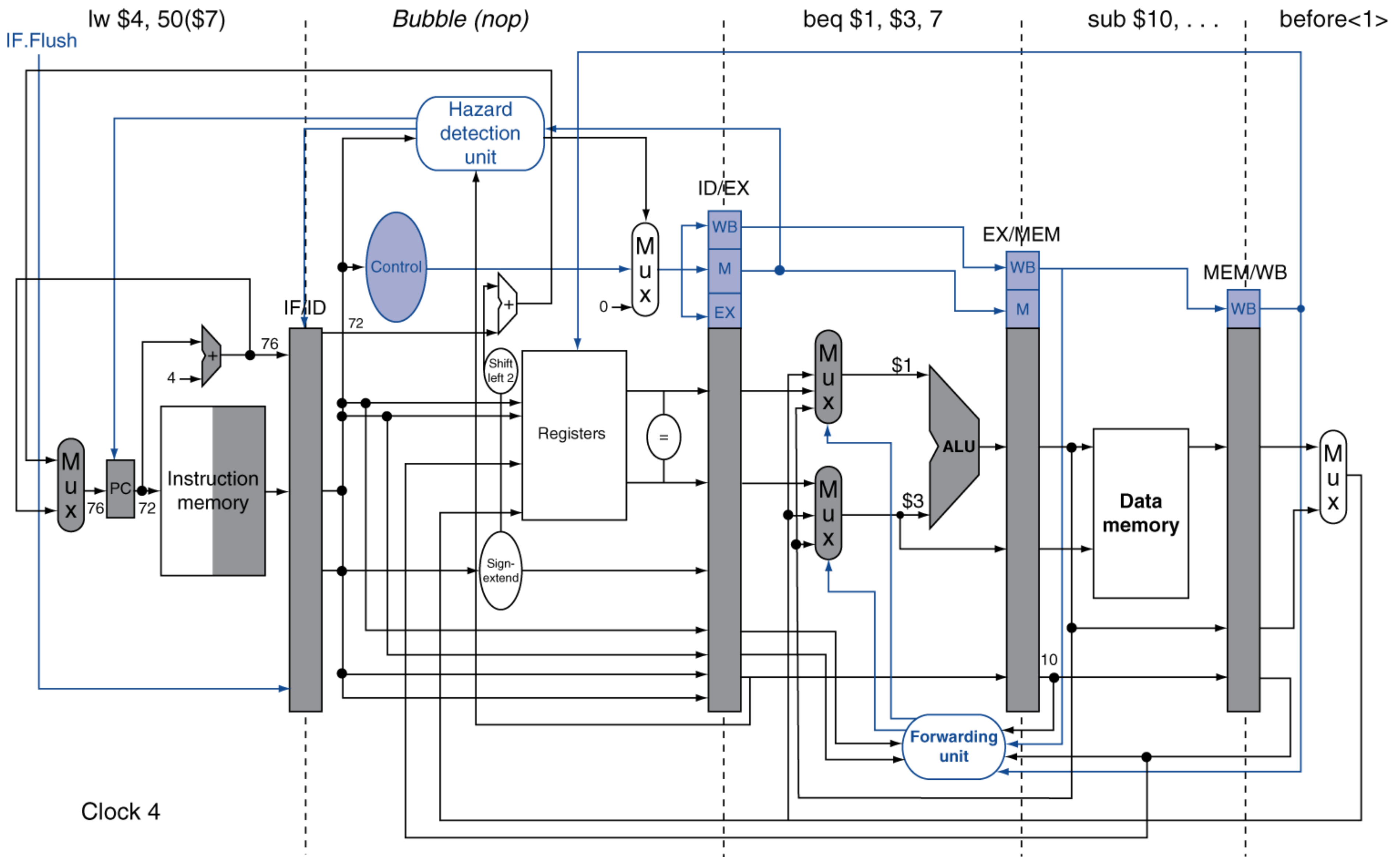


Παράδειγμα Λανθασμένης Πρόβλεψης

```
36:  sub  $10, $4, $8
40:  beq  $1,  $3, 7
44:  and  $12, $2, $5
 ...
72:  lw   $4, 50($7)
```



Παράδειγμα Λανθασμένης Πρόβλεψης

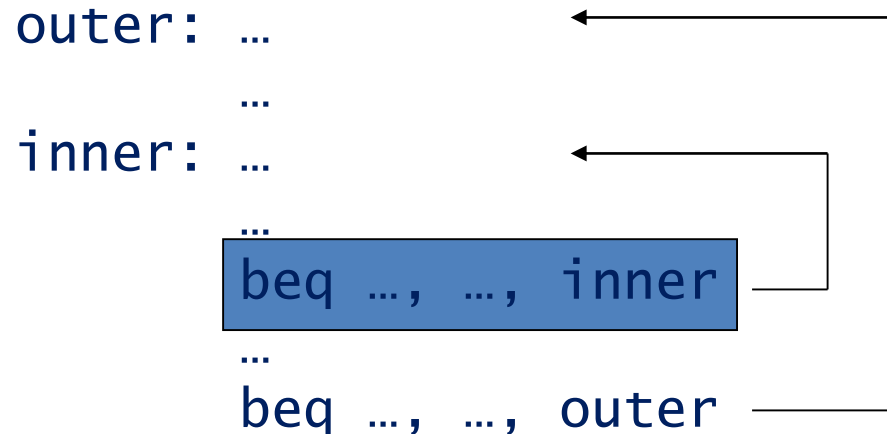


Δυναμική Πρόβλεψη Διακλάδωσης

- Στους σύγχρονους επεξεργαστές με μεγάλο αριθμό σταδίων διοχέτευσης (~20), ο αριθμός των κύκλων μηχανής που πληρώνουμε για λάθος πρόβλεψης διακλάδωσης είναι πολύ μεγαλύτερος.
  - Σημαντικός παράγοντας για την χαμηλή απόδοση του συστήματος.
- Η δυναμική πρόβλεψη της κατεύθυνσης μιας διακλάδωσης προσπαθεί να “μάθει” από την συμπεριφορά της διακλάδωσης σε προηγούμενες εκτελέσεις.

Δυναμική Πρόβλεψη Διακλάδωσης
Πρόβλεψη 1-bit

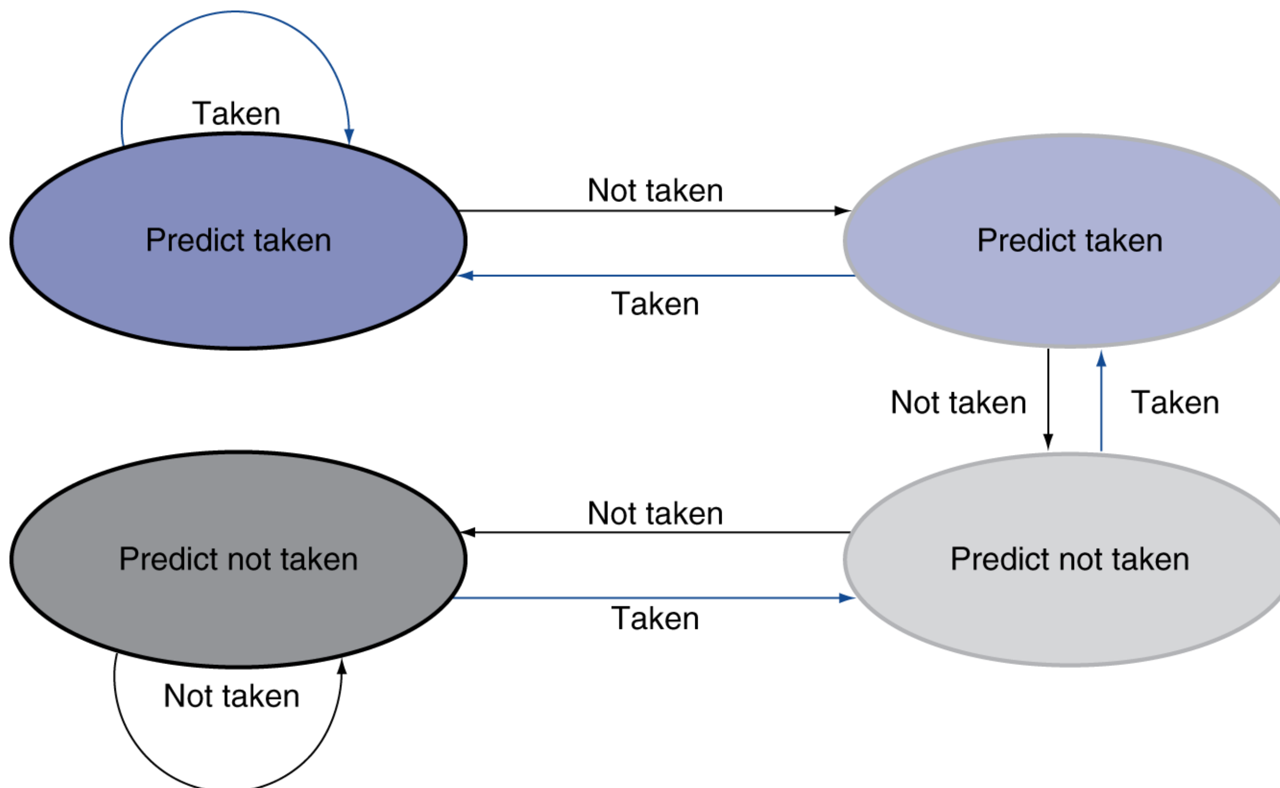
- Ο πιο απλός αλγόριθμος δυναμικής πρόβλεψης: προέβλεψε ότι η διακλάδωση θα συμπεριφερθεί όπως ακριβώς και στην αμέσως προηγούμενη εκτέλεση της.



- Απλή αλλά ασταθής λύση.
- Στο παραπάνω κώδικα η εντολή **beq** του inner loop θα προβλεφθεί λάθος 2 φορές για κάθε εκτέλεση του εξωτερικού loop.

Πρόβλεψη 2-bit

- Χρησιμοποιώντας μια μηχανή πεπερασμένων καταστάσεων μπορούμε να αποφύγουμε να αλλάζουμε απόφαση την πρώτη φορά που έχουμε λανθασμένη πρόβλεψη



Υπολογισμός Διεύθυνσης Διακλάδωσης

- Εκτός από την πρόβλεψη για το εάν η διακλάδωση θα είναι TAKEN ή NOT TAKEN, πρέπει να υπολογίσουμε και την διεύθυνση προορισμού.
- Branch target buffer (προσωρινή μνήμη προορισμού διακλάδωσης)
  – Μνήμη cache (μνήμες caches στο επόμενο κεφ.!) που αποθηκεύει μόνο εντολές διακλάδωσης , πληροφορίες σχετικά με την προηγούμενη συμπεριφορά της εντολής διακλάδωσης και την διεύθυνση προορισμού.
  – Κάθε φορά που είναι να φέρουμε μια καινούργια εντολή από την μνήμη, ελέγχουμε ταυτόχρονα και τον branch target buffer

    Εάν βρούμε την εντολή διακλάδωσης εκεί, και η διακλάδωση είναι TAKEN, τότε μπορούμε να φέρουμε την νέα εντολή από την διεύθυνση προορισμού που μας δείχνει η διακλάδωση

Αποτίμηση της Πρόβλεψης Διακλάδωσης

- Οι εντολές διακλάδωσης είναι ο κυριότερος ένοχος μειωμένης απόδοσης του συστήματος (μετά ίσως τις εντολές προσπέλασης μνήμης) .
  - Σε επεξεργαστές υψηλής απόδοσης (high performance processors), η λανθασμένη πρόβλεψη του branch επιφέρει πολλούς κύκλους ποινής.
  - Πολλές εντολές θα πρέπει να γίνουν flush σε μια τέτοια περίπτωση.
- Όλοι οι μοντέρνοι επεξεργαστές χρησιμοποιούν δυναμική πρόβλεψη διακλαδώσεων.
  - Ακριβείς προβλέψεις (>95%) της κατεύθυνσης διακλάδωσης είναι πάρα πολύ σημαντικές για καλή απόδοση ενός επεξεργαστικού συστήματος υψηλής απόδοσης.
  - Για αυτό και όλες αυτές οι CPUs χρησιμοποιούν πολύπλοκα συστήματα πρόβλεψης διακλάδωσης που “μαθαίνουν” την συμπεριφορά της εντολής κατά την διάρκεια εκτέλεσης του προγράμματος.
  - Πολύ σημαντικό ερευνητικό θέμα τα τελευταία 20 χρόνια.